<u>CS3283</u>

Media Technology Project I

AY 2014/2015 Sem I

Project on PPCDL

Progress Report 4

Mentor: Associate Professor Stephane Bressan

Terence Then Rei Jie (A0094682U)

Hans Adrian (A0100032M)

Christopher Andy Weidya (A0099568A)

Ko Wan Ling (A0100729M)

Tan Zheng Jie Matthew (A0101810A)

Table of Contents

1. Introduction

1.1. About PPCDL

The Powered Pleasure Craft Driving License (PPCDL) is required for the driving of powered pleasure crafts within Singapore's port limits. Obtaining a pass in both the theory and practical tests is necessary to get this license. With a license, one is allowed to operate powered pleasure crafts of length not more than 24 metres (excluding the propeller length).

1.1.1. PPCDL Test Structure

The PPCDL test is similar to the Singapore's driving license test in that they both comprise of theory and practical tests. The theory test consists of 30 Multiple Choice Questions (MCQ). A minimum of 26 out of 30 marks is required in order to pass the theory test. The questions may be factual (e.g. how long a battery charge of emergency lamp lasts) or situational (e.g. given a picture of three boats moving to the same point, which direction they should move next to avoid collision). On the other hand, the practical test requires test takers to show their boat handling skills. Both tests are carried out by the Singapore Maritime Academy (SMA) at Poly Marina Singapore Polytechnic.

1.1.2. PPCDL Theory Test Syllabus

Test takers will be examined on these different aspects of knowledge in driving powered pleasure craft during the theory test:

- Nautical Terms
- Boat Equipments and Usage
- Emergency procedures and prevention
- Aids to Navigation - Buoys, Beacons and Lighthouses
- Chart work and passage planning
- Tides and meteorology
- International Regulations for preventing collisions at sea (COLREG)
- Maritime and Port Authority of Singapore (MPA) regulations & Port Marine Circulars

- Taking off, berthing and anchoring
- Others (geographical knowledge, ropework , communications, etc.)

1.1.3. PPCDL Practical Test Syllabus

Test takers will be tested on their ability to practice safe powered pleasure craft driving. They may be tested practically on the boat or orally by the examiners. They will be tested on the aspects below:

- Preparation to get underway, such as equipment and passenger check
- Starting engines
- Unberthing, i.e. leaving the dock
- Maneuvering the boat
- Performing correct actions on a simulation of man-overboard
- Buoys and lighthouses
- Berthing, i.e. getting back to the dock
- Collision prevention
- Recognition of lights and shapes
- Action to be taken on certain events such as restricted visibilities and sound signal
- Other severe improper actions

1.2. About Boatmate

Boatmate is an application which puts the learning process of PPCDL tests into a game. It primarily runs offline. Our target users would be people who are planning to take the PPCDL examination. The aim of the application is to become a learning tool for PPCDL test takers in order to pass the theory and practical test. By making use of game mechanics and incorporating knowledge-based questions into the gameplay, Boatmate aims to enhance the learning experience of our target users, giving them a fresh perspective and a fun way to acquire knowledge regarding driving powered pleasure crafts and subsequently pass their license test.

1.3. User Types

Users of our system can be categorised into two different groups, namely:

- **Test Takers** - They are people who are preparing themselves to take the PPCDL tests.

- **Game Content Managers** - They are in charge of Boatmate’s question database and other interactive game content. Managing the question database involves the addition, modification and removal of questions and their related information (such as answers, explanations, etc.). Managing other interactive game content includes adding, modifying and removing media content on a level, changing the boats’ position in a game level, managing checkpoints and so on. Content managers are people who are qualified to teach PPCDL materials, such as certified PPCDL instructor.

2. User Requirements

2.1. Process of getting User Requirements

2.1.1. Meetings with Associate Professor Stephane Bressan

Associate Professor Stephane Bressan previously attended training lessons to prepare for the PPCDL test. He highlighted some problems he experienced in the process, namely the lack of free resources and the relative inconvenience of accessing these resources. These issues helped us to craft some essential user requirements which will be presented in the following section.

2.1.2. Online Research

Further research was conducted online gather more information on the test criteria, such as the examination syllabus and the test structure. The syllabus comprises of several categories concerned with different aspects of driving a boat. Such categories include navigation, safety and emergency, etc[1]. Some of the research result is summarized in Section 1.1 - About PPCDL.

2.1.3. Interviews and On-site Research

An interview was held with Captain Fadil, a PPCDL instructor at Poly Marina Singapore Polytechnic. He shared with us common problems that test takers face during the theory and practical tests. He recommended the usage of interactive elements in the game to address these areas. He also reviewed our first Boatmate prototype and a detailed discussion on this is found in Section 9 - Evaluation.

In addition to our meeting with Captain Fadil, we hope to conduct more interviews with potential users of Boatmate and other relevant authorities (namely the Maritime Port Authority of Singapore), such as PPCDL examiners. This helps us to clarify, refine and re-scope our user requirements where necessary.

[1] Refer to Section 1.1.2 - PPCDL Syllabus

2.2. Feature Lists for User Requirements

2.2.1. Feature List for Test Takers

Feature Lists	Priority
Test takers are able to learn and practice PPCDL concepts using Boatmate.	
● Test takers can practice questions in Boatmate that are relevant to the PPCDL test.	High
● Test takers can practice practical boat handling skills for different scenarios tested in the practical test.	High
● Test takers can receive proper feedback and explanation if they answer questions or handle the boat incorrectly.	High
● Test takers can perform an MCQ test that is similar to the format of the theory test.	High
Test takers can use Boatmate conveniently.	
● Test takers can access Boatmate on different platforms, notably web platform and mobile platform.	High
● Test takers can use Boatmate with keyboard, mouse and/or touch input depending on the device's input availability.	High
● Test takers can use Boatmate even if internet connection is weak or unavailable.	High
Test takers can track their learning process.	

● Test takers are able to identify categories of questions/tasks they are weak in through scores breakdown displayed in the statistics.	Medium
Test takers can receive targeted learning experience.	
● Test takers can choose to focus on learning specific parts of the syllabus.	High
● Test takers are able learn through various multimedia channels such as audio and video.	Medium
● Test takers can receive audio feedback when answering questions or practicing boat handling skills.	Low

2.2.2. Feature List for Game Content Managers

Feature Lists	**Priority**
Content managers can contribute content to Boatmate.	
● Content managers can add, modify and delete questions and answers that test takers receive on Boatmate.	High
● Content managers can add, modify and delete scene data in Boatmate (apart from default scenes).	High

3. System Requirements

Looking at our user requirements for the two target users, we find that the needs of a game content manager differ from a test taker's. As such, we decided to develop two applications. One of them is the game itself, which we refer to as "Boatmate" throughout the rest of this document. The other is an application for game content managers to change question database or other game content.

3.1. Use Cases

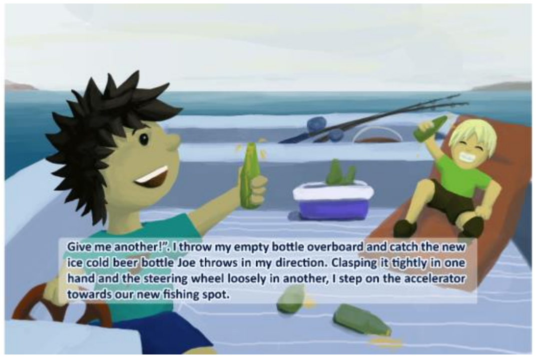
Essential cases that aid in understanding how Boatmate works are as shown:

Use case: U01 - Test taker plays Boatmate for the first time

Actor: Test taker

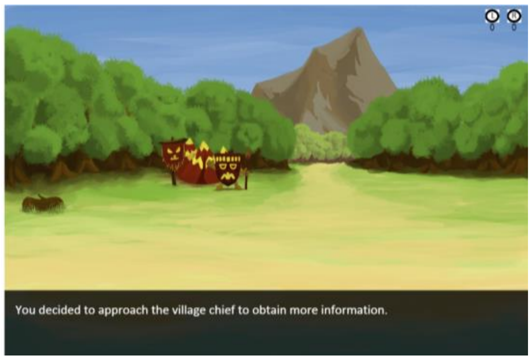
Preconditions: -

1. Test taker opens Boatmate.
2. If there is internet connection, Boatmate prompts user to update. Test taker can then choose whether to update the system, or not.
3. Test taker clicks "Start".
4. Boatmate shows the first cutscene.

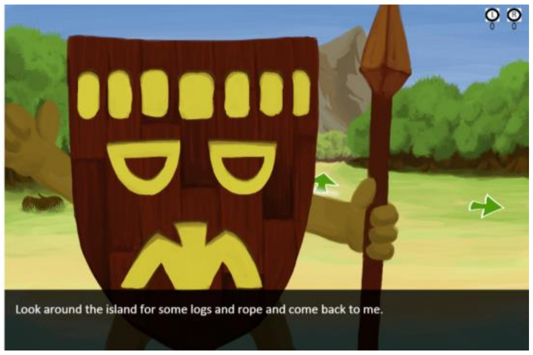


Give me another!". I throw my empty bottle overboard and catch the new ice cold beer bottle Joe throws in my direction. Clasping it tightly in one hand and the steering wheel loosely in another, I step on the accelerator towards our new fishing spot.

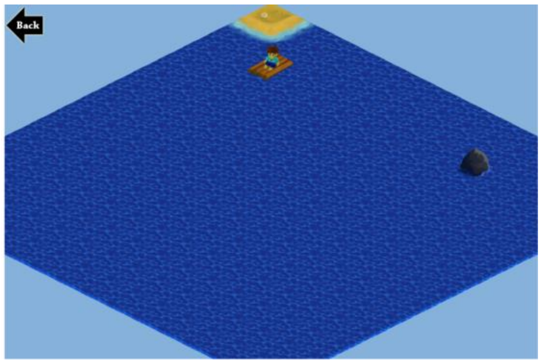
5. Boatmate shows the first set of instructions.


You decided to approach the village chief to obtain more information.

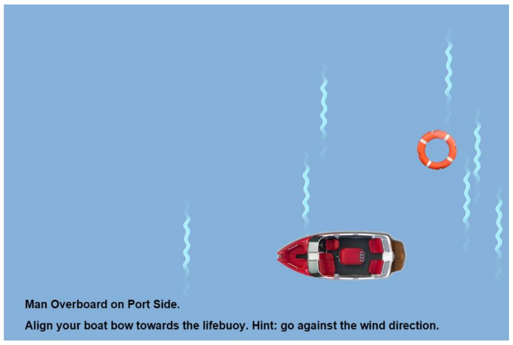
6. Test taker clicks on Non-Player Character (NPC)according to instructions.
7. Boatmate shows next set of instructions (task).


Look around the island for some logs and rope and come back to me.

8. Test taker completes the task given (Use case U03).
9. Boatmate shows the world map.



10. Test taker selects a tile to navigate to.
11. Boatmate presents test taker with a “Man Overboard!” mini-game.


Man Overboard on Port Side.

Align your boat bow towards the lifebuoy. Hint: go against the wind direction.

12. Test taker clears the mini-game.
13. Test taker exits Boatmate.

Use case: U02 - Test taker progresses through task in stage 1

Actor: Test taker

Preconditions: -

1. Test taker obtains 2 logs from exploring and clicking logs on the ground.
    a. Test taker clicks log on the ground.
    b. Boatmate removes log from scene and updates log count on screen.
2. Test taker clicks right arrow on the scene.
3. Boatmate displays east stage.
4. Test taker clicks on MCQ NPC and passes his MCQ test.
    a. Boatmate shows question on screen.
    b. Test taker chooses an answer.
        i. If test taker selects wrong, Boatmate explains why it is incorrect.
    c. Repeat a-c until all questions are done.
    d. Boatmate shows results.
    e. Test taker clicks “Okay”.
5. Test taker clicks left arrow on the scene.
6. Boatmate displays middle stage.
7. Test taker clicks on NPC in middle stage.

8. Boatmate instructs test taker to go to wreckage at east stage.
9. Test taker navigates to east stage and clicks wreckage.
10. Boatmate shows a screen of equipments.
11. Test taker selects required equipments.
    a. Test taker selects an equipment.
    b. Test taker drags equipment to either the “in box” or the “out box”.
    c. Test taker selects “Check”.
        i. If equipments are incorrect, repeat from a.

Use case: U03 - Adding a new MCQ quiz in an island checkpoint on Boatmate’s level editor

Actor: Game Content Manager

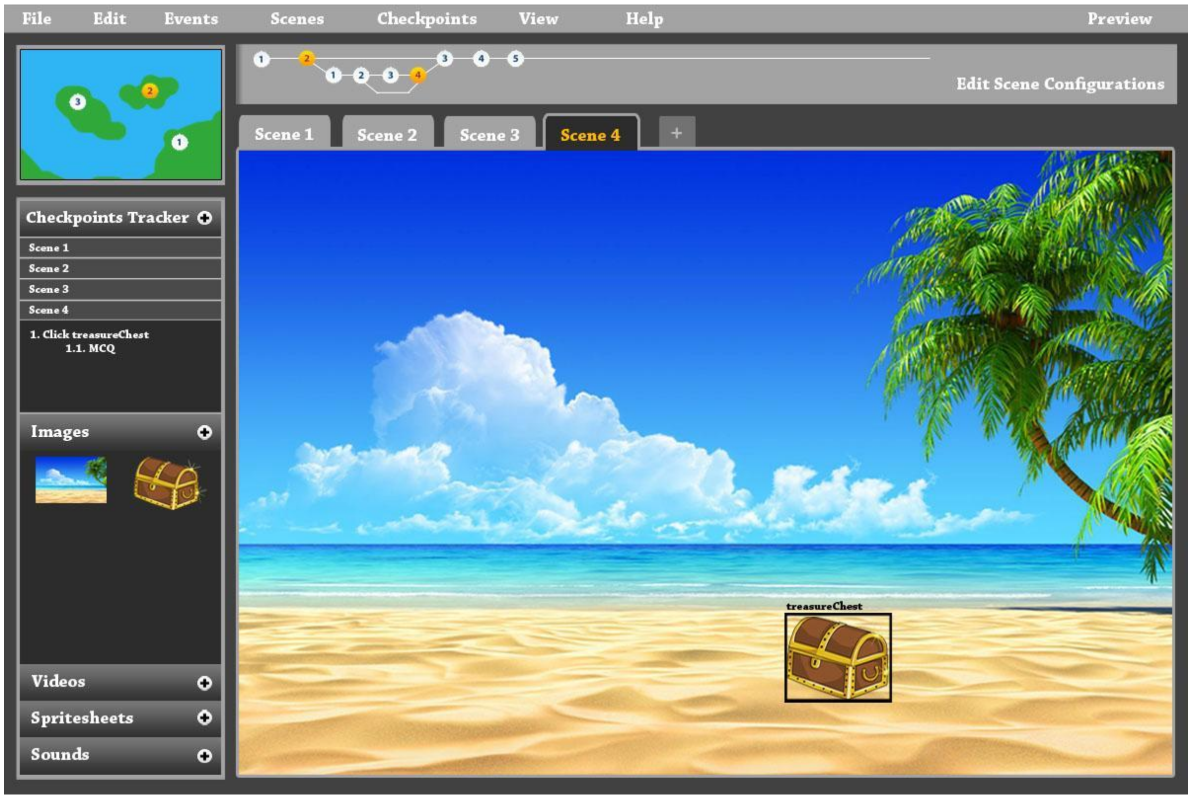
Preconditions: -

Situation: Actor is performing actions on server machine

1. Game content manager uploads a new image.
2. Game content manager drags uploaded image into scene shown in the main screen of the level editor.
3. Game content manager keys in the name of the added image in response to a prompt by the level editor.
4. Game content manager double clicks on added image to mark it as selected.
5. Game content manager clicks on “Events” in the menu bar and clicks on “Add MCQ Quiz”.
6. Level editor shows a popup with default list of questions pertaining to the category of the island checkpoint.
7. Game content manager selects option to customize questions.
8. Level editor shows a list of questions.
9. Game content manager filters and sorts the questions and checks those that he wants to add.

10. Game content manager clicks "Ok" once he is done with the selection.
11. Game content manager clicks on "Preview" in the menu bar and is able to playtest his newly added MCQ quiz.

A mockup of the level editor GUI is shown below for easier visualization.



3.2. List of System Functional Features

Feature Lists	Priority
Boatmate provides interactive ways for test takers to learn their PPCDL concepts	
● Boatmate is able to provide a multitude of question types to train test takers on their theory concepts.	High
● Boatmate provides interactive simulations to train the test takers on their theory concepts.	High
● Boatmate provides interactive simulations to train the test takers on their practical boat handling skills.	High
Boatmate has an extensive database of questions from PPCDL tests.	
● Boatmate's question bank database is able to store questions, answers, explanations, topic and links to multimedia files.	High
● Boatmate allows for selection of category (content) of questions for test takers to attempt.	High
● Boatmate is able to retrieve a set of questions from the question database given a specific number and topic.	High
Game content in Boatmate is easily extensible and changeable.	
● Boatmate has an interface for game content managers to easily modify the question's data (questions, answer options, correct answers and explanations) by filling in text fields in a	High

form.	
● Boatmate provides a level editor interface for game content managers to easily alter game content, such as setting up new checkpoints or controlling the appearance of random events.	High
Boatmate will keep track of test takers' personal progress.	
● Boatmate is able save test taker's progress in the game.	High
● Boatmate is able to store game statistics in database after the test taker is done with a set of questions.	High
● Boatmate is able to display game statistics to the test taker.	High
● Boatmate is able to synchronize test taker's progress cross-platform.	Low
Boatmate allows for customization in test taker's interface.	
● Boatmate can have sound muted by test takers.	Medium
Boatmate caters for different learning groups.	
● Boatmate can provide test takers with audio feedback specific to task or question.	Low

3.3. List of System Non-Functional Features

Feature Lists	Priority
Boatmate is convenient and easy to use.	
● Boatmate's test taker's interface runs in web and mobile	High

platform[2], i.e. run with conventional input (keyboard and mouse) and modern input (touch, accelerometer).	
● Boatmate can be launched without the need for an Internet connection.	High
● Boatmate has a responsive design.	Medium
Boatmate is efficient in its basic operations.	
● Each question will load in under 0.5 second.	High
● Boatmate will respond to a user's action in 0.7 seconds.	High
● Boatmate's assets (pictures, videos, etc.) will load in under 5 seconds.	Low
Boatmate is kept secure to ensure fairness amongst test takers.	
● Boatmate prevents test takers from prematurely viewing answers via debugging.	Low

[2] The system runs using HTML5, making it highly portable and accessible between platforms. More information can be found in Section 7.

4. Game Design

In this section, we will elaborate on the motivation that we make Boatmate as a game as well as how Boatmate's game is designed.

4.1. Motivation

Boatmate aims to assist PPCDL test takers to prepare for the actual PPCDL. In that regard, the game should allow the test taker to familiarise himself with the questions asked during the theory test as well as recognise what needs to be done during the practical test in order to pass. To achieve this, Boatmate will function as an edutainment game employing Game-Based Learning type of gameplay. Boatmate will also utilise various teaching strategies while leveraging on the interactive capability of games to keep the test takers interested.

4.1.1. About Game-Based Learning

Game-Based Learning (GBL) is a type of game that has well-defined learning outcomes. Generally, game based learning is designed to balance subject matter with gameplay and the ability of the player to retain and apply said subject matter to the real world.

GBL is used in order to be effective yet engaging in helping test takers to improve their knowledge. GBL has been proven to keep learners motivated to learn because of the sense of challenge it provides to themselves. Games with an accompanying, relatable storyline allows learners to immerse themselves in the experience better as they can empathizing with the character in the game. As playing games are voluntary, this allows us to tap on to their interest in the subject and keep them playing Boatmate.

GBL is considered better at teaching test takers in that it removes the fear of failure from traditional methods of teaching. In school, homework is considered more of an assessment rather than a learning opportunity. Homework asks the question of "do you know the solution?" rather than "can you solve this problem?". Using GBL, questions can be asked in a variety of manners instead of simply seeking the final answer.

Another comparison is that in school, students complete an assignment aiming for the highest possible grade, but are brought down by their mistakes, causing a reinforcing feedback loop for failure. This is very demotivational to the students. However, by making use of GBL, we can reward test takers as they complete tasks instead, removing this stigma of traditional education methods. It is also noted by studies that progress encourages even more progress by people. By rewarding learners in things they do, we can keep them wanting to do more and hence induce a habit to play our game.

Through such reasoning, Boatmate will make use of the mentioned game elements, such as a storyline, different types of questions and rewards for completing tasks and objectives.

4.1.2. Learning Strategies

It is known that a learner's preferred learning mode has significant influence on their behavior and learning and should be matched with appropriate learning strategies. VARK stands for Visual, Aural, Read/write and Kinesthetic[3]. They refer to the different sensory modalities in which people utilise in learning. People are typically inclined towards one of these four strategies (visual) but sometimes they are bimodal and have equal preference for two.

As the target users of Boatmate comprises of people of all backgrounds, it will be beneficial that our gameplay fits their learning styles. While not all parts of the game will suit every individual, there should be flexibility for the individual to choose his preferred gameplay, especially where it matters i.e. the learning portions. In that regard, the game will be set up with stages parallel to each other such that the user can choose a stage based on his preferred learning strategy.

[3] http://www.vark-learn.com/english/page.asp?p=categories
http://www.academia.edu/4516457/Different_Perspectives_of_Learning_Styles_from_VARK_Model

4.2. Game Storyline

Boatmate uses a narratological approach in order to create a narrative desire for test takers to continue the game. The story begins as the test taker is out fishing with his friend, Joe, and crash after they disregarded a sign they passed by. Shipwrecked on an island, the test taker has to build a new boat in order to find Joe and return home.

To do so, he has to perform tasks in exchange for materials he may use to upgrade his means of travel. An example of this would be collecting logs and rope from local islanders in order to build a simple raft. The test taker will have to travel to various different nearby islands to gather all the materials to build a boat sturdy enough to get him and Joe back home.

After completing the final boat and sailing back home, he flashbacks to when he was shipwrecked and has to overcome his fears through the knowledge he has gained while constructing his new boat.

4.3. Gameplay

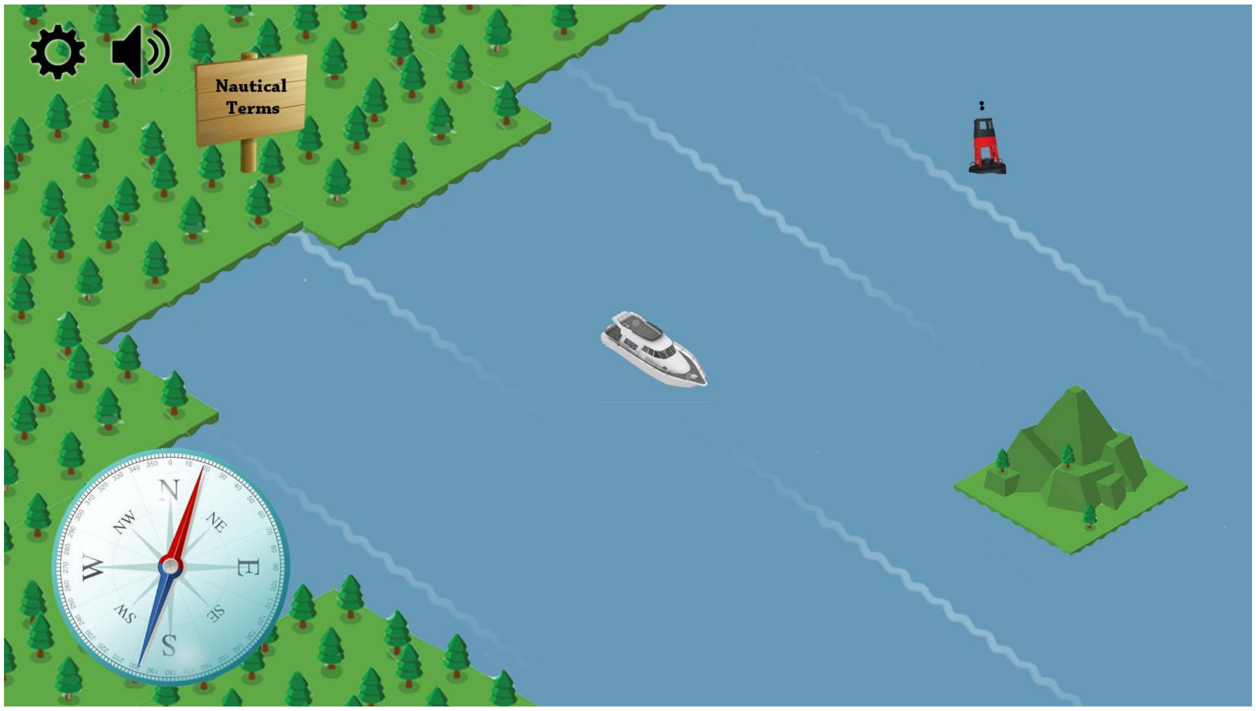
Boatmate consists of several layers of gameplay. The general gameplay is relatively railroaded, requiring the test taker to follow a sequence of checkpoints to reach the end of the game. This guarantees a comparatively standardised learning experience[4] for all test takers. The gaming experience, however, varies for each test taker as they have the freedom to choose how they complete their objectives. Boatmate's gameplay can be divided into three major components: 1) Map Navigation, 2) Exploring Islands and 3) Completing Tasks.

4.3.1. Map Navigation

Map navigation is where the test taker moves the character's boat across a map (see below). Using a mobile device or a desktop, this is done by pressing or clicking at the target location, respectively. The map consists of a sea with 10 islands. Each of these islands besides the first one represents a major PPCDL concept which will be tested there. The first island serves as introduction to the game.

[4] Dimensions in Educational Game Design by Thomas Duus Henriksen

Since certain PPCDL concepts require foundational knowledge obtained from other PPCDL concepts, not all islands will be accessible at the start. This is achieved by the usage of map obstacles such as fogs and storms on the map to prevent test takers from reaching those islands. These islands will only be accessible once test taker has learnt the required concepts to understand concepts in the islands. Accessibility is attained by upgrading boat from currently accessible islands.



Mockup of navigation in Boatmate, where test takers move between islands

While travelling on the map, random events may be triggered. These random events add an element of unpredictability to Boatmate and increases the dynamicity of the game. Random events cover a wide range of emergency situations that a test taker will encounter while driving a real boat. These include scenarios such as encountering a sinking ship, capsizing or a man being overboard. During these random events, the test taker will play a mini-game related to the random event he or she has encountered. These mini-games simulate emergency conditions and require the test taker to perform the necessary actions to clear it. In the case of coming across a sinking ship, the mini game will be about rescuing the ship's passengers with simulated boat control. The type of the random event is selected based on the test taker's progress in the game. This is because some random events require prior knowledge about collision rules and hence will not be given to the user before he completes that island.

4.3.2. Exploring Islands

Each island represents a topic in the PPCDL syllabus. Upon entering an island, Boatmate will display an inland view. An island can have two or more separate inland view to represent different parts of the island. In each of these islands, the test taker's main objective is to gather materials. These materials are then used to upgrade his boat, which is necessary to progress in Boatmate. Materials can be gathered by interacting (clicking) with on-screen elements which consist of NPCs and scene objects.

During this process, the test taker will be educated on PPCDL concepts through mini-games or NPC interactions. Once the test taker has sufficient materials to leave the island, he should be well versed in the PPCDL topic assigned to the island. The island is always available for future revisits.

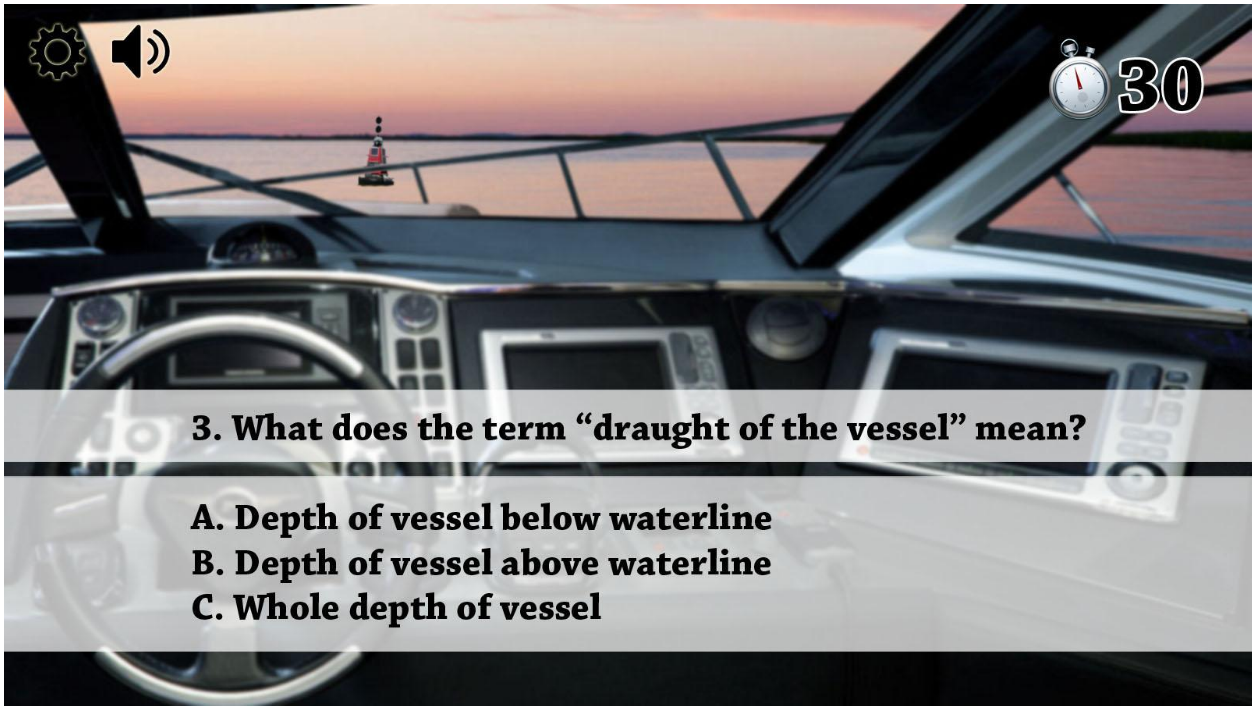
4.3.3. Completing Tasks

There is a large collection of tasks that a test taker can complete to obtain his or her desired materials. These tasks include MCQs (see below), drag and drop type questions and other mini games. Each of these tasks is designed with the learning strategies and GBL mentioned above in mind. A few of these tasks are described in detail below.

MCQ Tasks

MCQ tasks will quiz a test taker on the category the test taker is currently learning. In this task, the test taker is presented with a series of MCQ questions. If the test taker chooses the wrong option, an explanation is provided explaining why that option is wrong and they have a second chance to get it right. If he or she answers incorrectly once again, the MCQ task is restarted. This encourages the test taker to get the correct answers by understanding or research rather than wild guessing. Some test takers might be frustrated from this approach, but we want to prioritise and enforce their understanding of concepts through each MCQ task. The notion of scores is, however, not inherent in these MCQ tasks as they are formative in nature. After completing the MCQ task, the test taker is rewarded with the promised material.

Questions for the MCQ tasks can be accompanied by a pictures, audio or video clips which the question references. For example, a picture of a boat may be shown with an arrow pointing to a specific part of it, with the question asking which part of the boat it is pointing to.



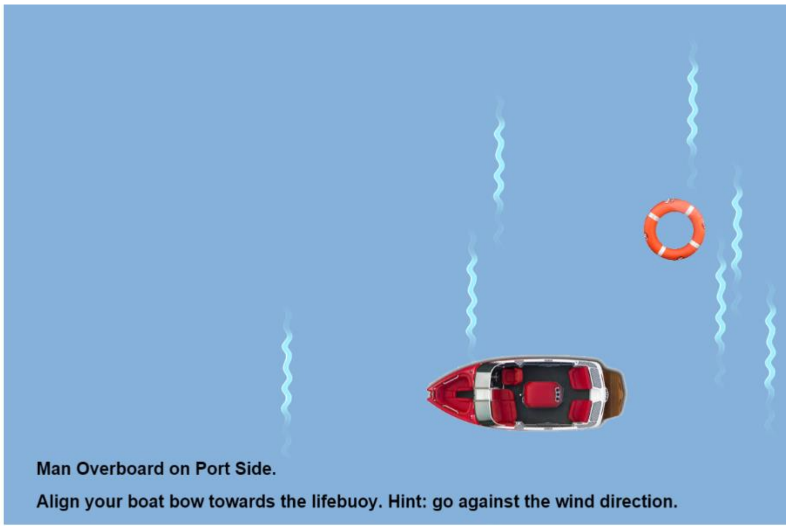
3. What does the term “draught of the vessel” mean?

A. Depth of vessel below waterline
B. Depth of vessel above waterline
C. Whole depth of vessel

Mockup of MCQ questions found in a task

Interactive Tasks

Interactive tasks span a large category of specialised tasks to complete. By performing specific actions in the game based on the question, the test taker can complete the task and earn the material specified.


Man Overboard on Port Side.
Align your boat bow towards the lifebuoy. Hint: go against the wind direction.

Mockup of Man OverBoard Interactive Task

An example of an interactive task would be the toggling of navigational lights. The question provides the test taker with a picture of a boat with all its navigational lights off and asks the test taker to turn on the lights required for if the boat was involved in fishing activity. The test taker would then click on the desired navigational light to toggle it and press the submit button to check if it is correct. If answered wrongly, the test taker will be given another attempt before a new task is presented to him.

Another example of an interactive task is the dragging and dropping of items to the right places. For example, the task may present the test taker with a box of 20 items and ask the test taker to remove the items that are not part of the safety requirements for the boat. The test taker has to drag the wrong items out before pressing the check button to see if it got it right or not. If answered wrongly, the number of misplaced items is reported to the test taker and another attempt is offered to him before the items are reshuffled once again.

There are a large number of other interactive tasks which are not written in detail here but each one of them tries to provide an alternative gameplay apart from the standard MCQ style in order to cater to the various learning strategies of our players.

4.4. Dynamic Game Experience

Every test taker is different in terms of their preferences and capabilities. As such, Boatmate is able to adapt according to the test taker’s profile. This is primarily done through the capture of statistics when the test taker performs tasks. Results of the tasks performed by the test taker are kept and analysed by category.

From the statistics, we can analyse both the strong and weak categories of a test taker's knowledge. Using this information, the game can adjust the material requirements needed for the next upgrade of the test taker's boat to require more from that island category. For example, if the test taker is detected to be poor in nautical terms, and the material obtained from the nautical term island is wood, the requirement to upgrade the test taker's boat will require three more wood than the norm.

Potentially, these statistics can be uploaded to the server for game content managers to evaluate on the game content as well as PPCDL content itself. If many players fail in certain categories, game content managers can investigate to find out the reason. Depending on the reason, he might want to add more questions of that category or change the question's wordings. Also, since game content managers are certified PPCDL instructors, they might want to check if they have covered those categories extensively enough on their lessons for the test takers to be able to pass.

5. Technical Details

5.1. Development Tools

Below are tools that we use for developing Boatmate.

5.1.1. Implementation Tools

- Javascript
- Phaser: HTML5 framework
- CocoonJS: mobile development framework
- MySQL: relational database management system

5.1.2. Project Management Tools

- Trello: https://trello.com/b/teeVkmpk/shipbros
- Github: https://github.com/HansNewbie/ShipBros

5.2. Implementation Progress

5.2.1. Completed Tasks in Semester 1

We have implemented the following in Boatmate:

- Start menu that links to the game
- Introductory story cutscenes
- Single island exploration and mini quest of collecting materials
- Man overboard random event that occurs in the navigation stage
- Safety equipment check that comes in the form of a drag and drop game
- Point to point movement in an isometric map for the navigation stage
- MCQ quiz with explanations and score
- Client side question processor
- Client side JSON parser (database DAO)

5.2.2. Tasks to be Completed in Semester 2

The following is a list of tasks that we intend to complete in the next semester. The tasks and their priorities are subject to change in the future if necessary.

Tasks of High Priority

- Create island checkpoints based on all topics in PPCDL
- Create additional game scenes, random events and mini games for each topic in PPCDL
- Create a separate section in the main Boatmate application that allows test takers to play customized islands made by content managers
- Implement basic questions manager that allows CRUD of questions in Boatmate
- Implement level editor with basic functionalities:
    - Creation of new checkpoints within an island
    - Upload of media content, such as images, videos and sounds
    - Alteration of multimedia content of game scenes (eg. replacing background images or spritesheets)
    - Customization and adding of MCQ questions to a checkpoint

Tasks of Medium Priority

- Implement test taker statistics features
- Implement level editor with intermediate functionalities:

    - Preview option before syncing changes to Boatmate clients
    - Basic alteration of interactive tasks (eg. specifying wind direction in a man overboard random event, setting chance of occurrence of random events)

Tasks of Low Priority

- Implement watercraft customization
- Implement level editor with advanced functionalities:
    - Navigation map customization
    - Extensive customization of interactive tasks and mini games, such as deciding the logic behind each body in the game or altering physics conditions in mini games

5.3. Design and Implementation Considerations

This section will further elaborate on the substantiation and implementation of interactive features in Boatmate. Consideration of the following reasonings are important in helping us decide on the specific frameworks and design decisions visible in our system architecture[5].

5.3.1. Cross-platform Application

Design: Boatmate is conveniently available on three main platforms (iOS, Android and web).

Technical Decision: HTML5 will be used to code Boatmate due to its high mobile optimization and cross-platform compatibility.

Justification/Explanation: As of 2013, both Android and iOS own over 94% of the global market share of smartphone operating systems[6]. Hence, targeting iOS and Android ensures that most test takers with a smartphone are able to run Boatmate. Test takers can also run Boatmate on any browser on a desktop if they do not have smartphones. This support for both web and mobile devices is achieved through the HTML5 framework. The improvement of HTML5 over its predecessors in its processing model and standardisation has resulted in its usability in almost every mobile device and web platform. Hence, its cross-platform compatibility along with new

[5] This will be discussed in Section 7.

[6] http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

application programming interfaces for web application allows Boatmate to smoothly implemented for iOS, Android and web devices.

5.3.2. Highly Interactive and Responsive Application

Design: Boatmate will have game elements such as interactive gameplay, storyline, mini games and exploration, which rely heavily on the fast handling of user inputs.

Technical Decision: Boatmate will use Phaser framework for its implementation, having its architecture design similar to that of typical game architecture.

Justification/Explanation: Boatmate at its core is an interactive game. Thus, we have modelled Boatmate's architecture similar to those used in games in order to handle user input and respond to them interactively. Boatmate's GUI is not simply a container that only displays images or text; it is capable of handling input and updating what to display on its own in addition to its standard graphics displaying function. This reduces additional overhead from repeatedly calling an external input parser or logic component.

The Phaser framework uses HTML5 technologies. It utilises Canvas renderers which uses less processing power on both web and mobile platforms compared to native renderers. This ensures that mobile devices, which typically has less computing capability than a desktop or laptop, can run Boatmate smoothly. Furthermore, Phaser also has a light-weight physics library which handles game elements such as collision detection in an efficient manner. Its capability to support multiple forms of user input allows Boatmate to respond to both touch and mouse and keyboard inputs. Game camera, sprites and various game elements are also in Phaser’s library. Phaser’s comprehensive support for game development and its smooth processing ability are therefore prime justifications for its usage.

5.3.3. Offline Capabilities

Design: Boatmate is accessible offline to make it highly accessible and convenient to use.

Technical Decision: CocoonJS will be used as a wrapper to convert Boatmate into native applications for iOS and Android as well as into a Chrome app for desktops.

Justification/Explanation: After the initial installation, Boatmate can essentially run offline. If internet connection is present, test taker will be prompted if they want to update the game content when he launches Boatmate. As such, test taker can proceed with the normal functionalities of Boatmate if there is no internet connection, except that the game will not use the latest questions or game content.

CocoonJS is HTML5 compliant and will be able to suitably function as a wrapper for Boatmate without losing or unintentionally changing the original functions implemented. Remote debugging of Boatmate on a mobile device is also possible with CocoonJS.

5.3.4. Extensibility of Database and Application

Design: A question bank supplemented with answers, explanations, question types and categories in different forms of multimedia will be implemented to support PPCDL’s vast amount of content. Boatmate will also allow game content managers to easily update or create new questions to accommodate for new PPCDL content in the future.

Technical Decision: We will use MySQL to implement the database of Boatmate as it achieves efficiency and scalability while maintaining integrity of data[7].

Justification/Explanation: MySQL’s cross platform support and effective storage procedures are the main reasons for its usage for Boatmate’s database. Since Boatmate has a large database due to PPCDL’s large knowledge base, MySQL’s scalability and self-healing data integrity is crucial to Boatmate.

[7] A sample of our planned question data format and attributes can thus be found in Section 7.

Game content managers, however, are not expected to fully understand low level database commands and implementations. Thus, a graphical user interface for easy adding, removal or update of questions will be implemented for their use. Furthermore, Boatmate's architecture separates work elements into comprehensible units, introducing high cohesion and low coupling for easy extension in the future.

5.4. Technical Focus

Boatmate focuses on the implementation of multimedia systems, specifically games. Boatmate will focus on the following areas: game user interface design, path finding, procedural content generation[8] and simulation.

5.4.1. Game User Interface Design

Boatmate is a multi-platform game and the considerations in making a game's user interface is rather different compared to any other software. Many user interface considerations are needed for several reasons. For one, controls for the game have to be intuitive for keyboard-mouse usage as well as touch controls. Another important consideration is the visual aesthetics. The game has to look good on both mobile devices and their PC browser counterpart, which have varying screen sizes.

5.4.2. Path Finding

Boatmate's navigation gameplay uses click-to-move controls for the watercraft movement from its initial position to the chosen tile. As this is a point to point movement, path finding is required in order for the boat to avoid impassable tiles, such as obstacles and islands, which may be on the direct path between the tiles.

5.4.3. Procedural Content Generation

Boatmate has many game stages which share the same gameplay but have different content. These differences are generally difference in text, images and image positions. For example, two MCQ stages can have different backgrounds, different character pictures and different questions. Another example would be two interactive questions using a top-down boat simulation. In this case, it may involve different number of watercrafts, different watercraft

[8] http://pcg.wikidot.com/

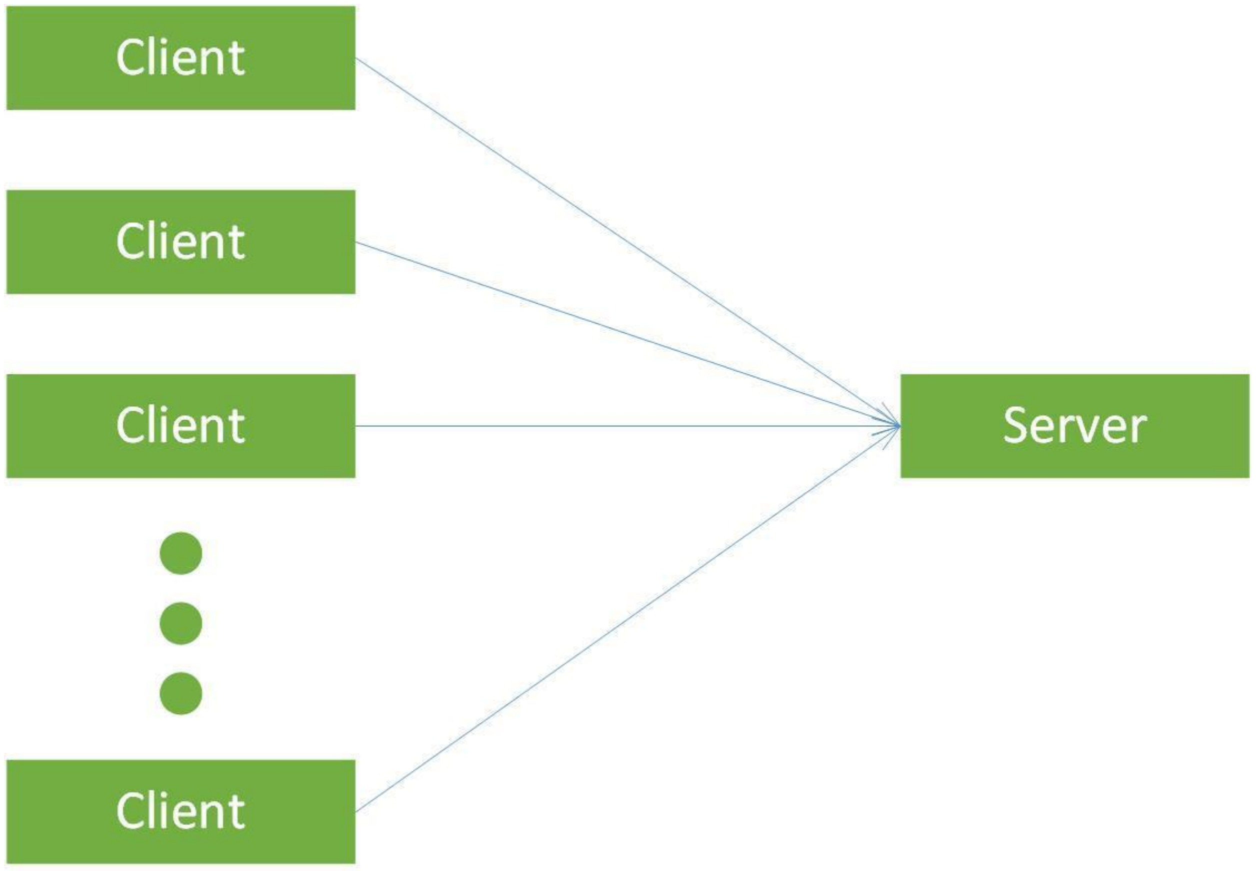
pictures and different watercraft positions and orientations. As such, Boatmate will be designed to be procedural in terms of content. This also potentially makes our Boatmate system extensible for other license test learning platforms such as car driving license tests or even the advanced PPCDL.

5.4.4. Simulation

Many mini games in Boatmate involves simulating a real life situation on the sea for the test taker to practice the sequence of actions that he needs to take. While Boatmate only stays on 2D graphics, we would like to simulate the physics of watercrafts on the water and watercraft control as much as possible to allow our test takers to have as real an experience as possible compare to the actual test.

6. System Architecture

Boatmate uses client-server architecture, as shown below.



A more in depth view of our architecture will be discussed in the following sections.

6.1. General System Architecture

Within the client and server side, layered architecture is used. In the software architecture diagram below, the layer is put across horizontally instead of vertically for ease of viewing due to our architecture design. Each layer will be given a different color:

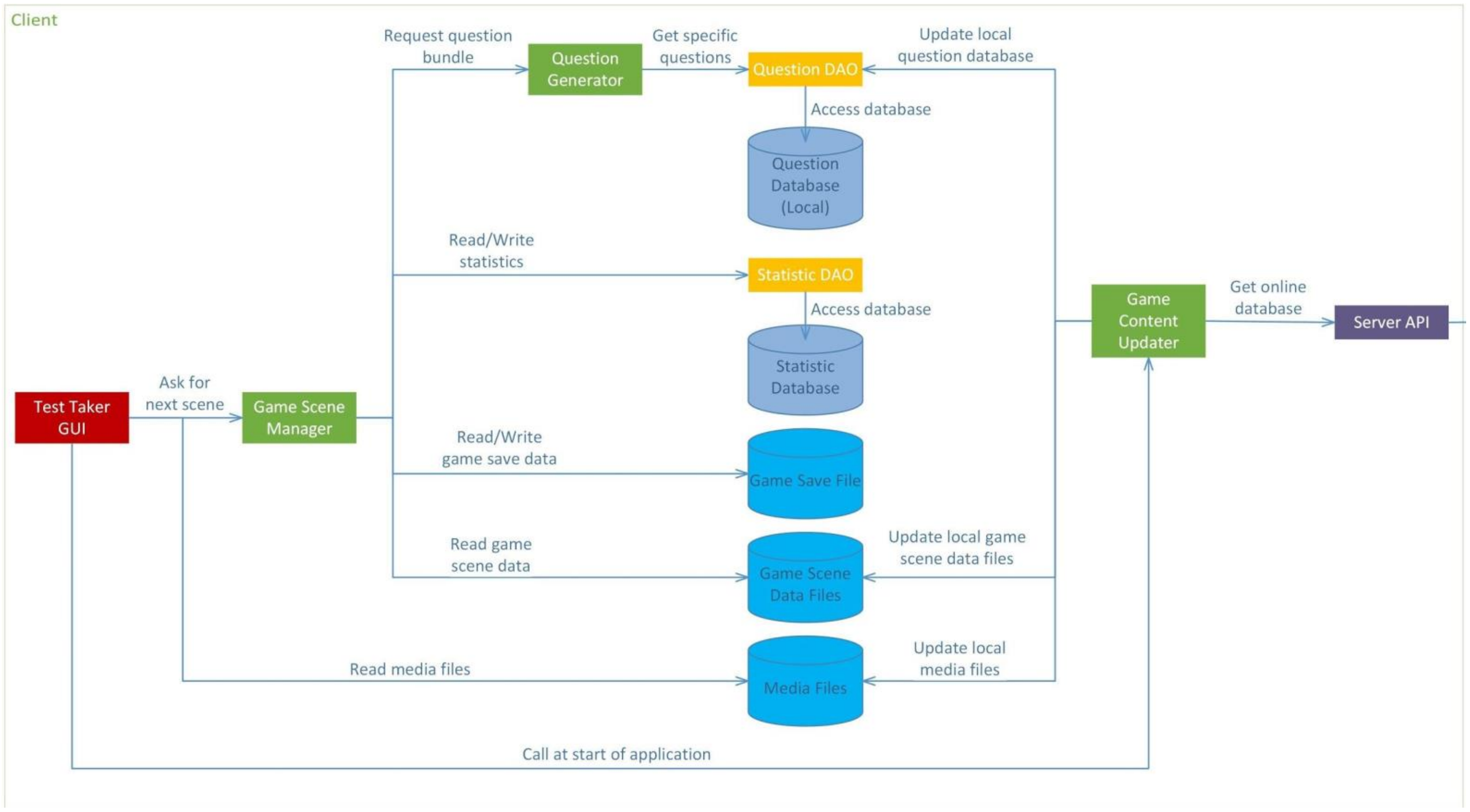
- Red: Graphical User Interface (GUI)
- Green: Logic
- Yellow: Database Access Object (DAO)
- Purple: Network interface
- Dark blue: Database storage

- Light blue: File storage

As Boatmate is written in JavaScript, JSON (JavaScript Object Notation) will used for the local databases as there are readily available libraries to access JSON in JavaScript.

6.1.1. Test Taker Client Side

We will cover in depth on the architecture used in the Client side in this section.



<u>Boatmate client architecture</u>

Name	Description
Test Taker GUI	Responsible for displaying the game and handling input. Being a game, it would be difficult to separate input handling and display in Boatmate. As such, both are handled by Test Taker GUI.

	Test Taker GUI will handle all input and interaction in a scene (example, as long as test taker is in boat navigation mode, Game Scene Manager will not be called). If another scene is needed, for example when a checkpoint is clicked or a task has been done, Test Taker GUI will call Game Scene Manager to get the next scene. Game Scene Manager will decide what scene is next. Phaser framework is used for Test Taker GUI. Input handling is done by Phaser framework.
Game Content Updater	Responsible for updating local (client-side) data. Game Content Updater will only be called once at the start of the application by Test Taker GUI. If there is internet connection, user will be prompted if they would like to update the game content. If user chooses yes, Game Content Updater will ask for current relevant data - currently question database, game scene data files and media files - in server (through Server API) and update (if there is any change) the respective local data through Question DAO. Otherwise, the application will proceed with the current content stored in the local file system and database.
Server API	A facade to the server. This separates the concerns of sending data to and receiving data from the server through the Internet.
Game Scene Manager	Responsible for providing game scenes to be displayed and handled by Test Taker GUI. Game Scene Manager will be highly coupled as it handles saving/retrieving of gameplay statistical data, saving/loading game save data and giving scene data -

	and questions if the scene requires it - for Test Taker GUI to display and handle. Game Scene Manager will not be selecting which question data to be played at the current stage; it will request Question Generator for a set of questions of specific type and category. It will then pass this set of questions to the GUI to be shown. Scene data includes file addresses for the scene and which type of scene. Types of scene are limited, depending on what has been developed into Boatmate. A new scene type will require new version of Boatmate.
Question Generator	Responsible for providing a set of questions of specific type and categories. Question Generator will get the requested category-type-combination questions from the local question database (through Question DAO), select some of them to be displayed, randomise the options to be shown with the correct answer recorded, and return them to Game Scene Manager as a bundle of questions.
Statistic DAO, Question DAO	Each of them is used to provide an interface to their respective databases. They support the single responsibility principle in the sense that other objects do not need to know the database specification. They also ease changing of database schema or choice of database system since the API they provide for object accessing database will not change.
Statistic Database	It stores test taker's gameplay statistics. Gameplay statistics include: number of times of test taker answering correctly and wrongly and the number of questions of a specific type-category-combination that the test taker answers correctly and

	wrongly. We are considering using JSON format for this.
Question Database	It stores the repository of questions. We are considering using JSON format for this.
Game Save File	It stores the game save data. Game save data includes test taker’s journey progress in the game and test taker’s ship customisation. We are considering using simple text file format for this.
Game Scene Data Files	It stores the possible scenes that Game Scene Manager can tell Test Taker GUI to display and handle. It may include some or all of these: file addresses, question type-topic combination, position of objects in scene - i.e. scene setting, next scene to display once the scene is completed.
Media Files	It stores the game’s media assets, such as videos, sprite images and sounds.

As stated above, an interaction that does not change scene will only involve Test Taker GUI. This means that Test Taker GUI will process user input and handle the input by itself in changing the state a scene is in, e.g. where the boat should be moved. This is to reduce overhead between reading input, parsing it and updating the game respectively. Also, Test Taker GUI will get media files such as images and audio directly instead of through Game Scene Manager because Game Scene Manager will not use these assets, thus cutting the man in the middle.
Question Generator is made into its own component so that Game Scene Manager does not need to handle which questions should be set for current MCQ gameplay and randomizing the answers. Game Scene Manager only requests for question type-topic combination and will pass the result to Test Taker GUI without needing to concern itself with question handling.

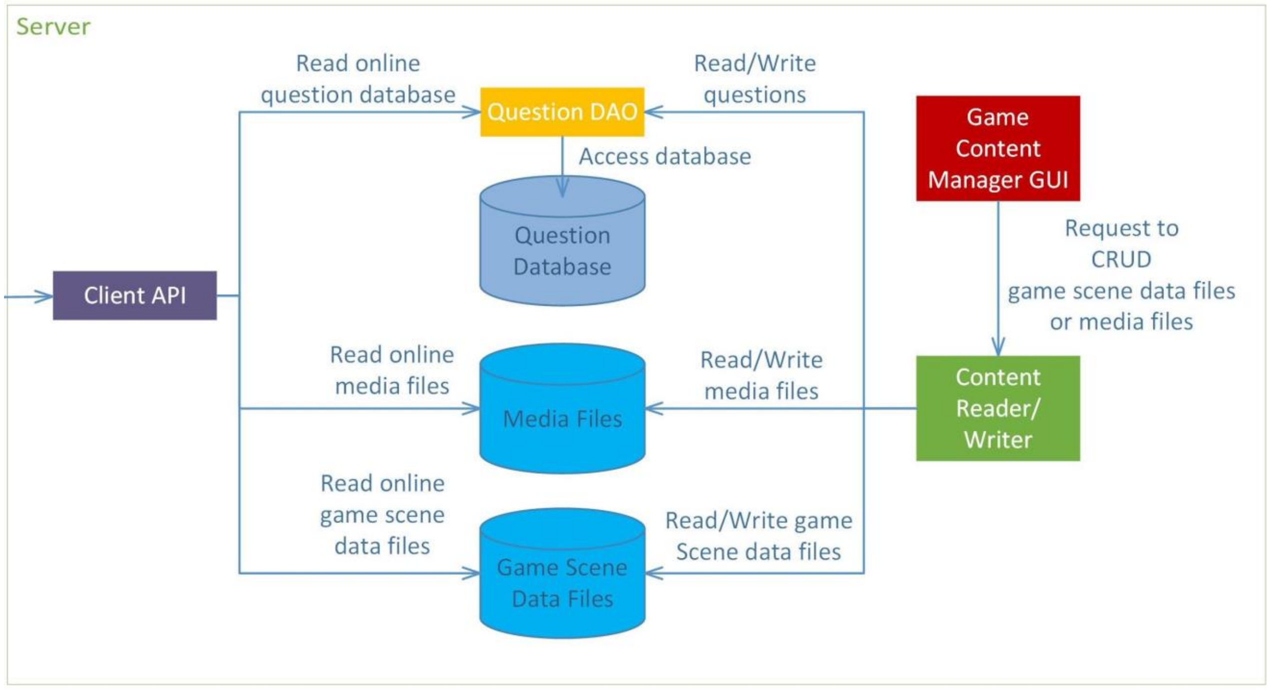
Game Scene has one of the highest coupling in our architecture. It is necessary since it is the component that glues all the other components. That being said, one concern might be on saving game data which will be handled by Game Scene Manager. While it is possible to have it handled by a separate component, it is more logical to have Game Scene Manager doing so. Firstly, having a component to handle saving the game data would means Game Scene Manager relaying the data from Test Taker GUI since the system would save the data after every scene change. Secondly, Game Scene Manager would know most part of a save data since save data involves many parameters of game scenes which Game Scene Manager does. Having a component to know these parameters again would result in code duplication.

Game Content Updater has a single responsibility of updating game content if internet connection is present. It is separated from Server API for extensibility. For example, we would like to sync statistic data or game save data with our server. We will need Server API but we are not updating. As such, we still have an updater and we will have a sync manager, and both are connected to Server API in connecting to the server.

We have two databases - one for statistic and one for question - because they essentially hold completely unrelated data. We have one DAO for each so that the APIs that components accessing the database will be the same regardless of the database implementation.

6.1.2. Server Side

We will cover in depth on the architecture used in the Server side in this section.



Boatmate server architecture

Game content managers will access the server machine in order to edit Boatmate's game content. Thus, login validations will apply at the operating system level and not at the application level. This essentially makes access to the Boatmate level editor restrictive, which supports the fact that game content managers are a very specialized small group of users.

Name	Description
Client API	A facade to the internet. This separates the concern of sending data to and receiving data from the client through The Internet.
Question DAO	It is used to provide an interface to their respective databases. It supports single responsibility principle in the sense that other objects do not need to know the database specification. It also eases changing of the database schema or choice of database system since the API it provides for object accessing database

	will not change.
Question Database	It stores the repository of questions. We are considering using MySQL for this.
Game Scene Data Files	It stores the possible scenes that Boatmate client can use. It may include some or all of these: file addresses, question type-topic combination, position of objects in scene - i.e. scene setting, next scene to display once the scene is completed.
Media Files Storage	It stores all media related content required for Boatmate, such as videos, sprite images and sounds.
Game Content Manager GUI	Responsible for displaying and handling input for level editor. It will call content reader/writer if game content manager loads a game scene, looking for existing assets or saves current game scene. Game scenes may be saved without being published. Phaser framework may be used to handle scene.
Content Reader/Writer	Responsible for reading and writing of game scene data files and media files. It will read game scene data files if game content manager would like to edit existing game scene. It will read media files if game content manager edits existing game scene (thus needing the assets of the game scene to be shown) and if game content manager would like to use existing media files for a new game scene. It will write media files if game content manager would like to add new assets to a game scene.

As with client architecture, Game Content Manager GUI manage more than just getting input and displaying information. It handles changes on its own as changes made on one scene - say boat - will not be saved until game content manager saves it. As such, one session utilises mostly graphic element and straightforward data that does not need calculation, and it would be easier to have GUI to handle all this on its own.

We still have Content Reader/Writer so that Game Content Manager GUI does not need to concern with the data storage separation for reading and writing.

6.2. Question Attributes

The server will maintain a questions database which can be updated by game content managers. The client can only update its local question database when it is connected to the internet.

The questions database is organised into 3 main tables being the Topics table, QuestionTypes table and Questions table.

<u>**Topics table**</u>
Topic: String
The topic contains a string which specifies which part of the syllabus does the question belongs to. More information on the PPCDL syllabus can be found on Section 1.1.2 - PPCDL Syllabus. There are a total of 10 different topics with which we can categorise our questions.
Topic ID: Int
The topic ID contains an integer that will represent that topic. It is the primary key for the table. Requests made to retrieve a set of questions of a certain topic will use the topic ID instead of the full topic name.
<u>**QuestionTypes table**</u>
Question Type: String
It contains a string which specifies which question type does the question belong to. Examples of question types are: MCQs and 'Drag and Drop'. Different question types require test takers to answer the question in different manners (e.g. MCQs require users to click whereas Drag and Drop requires users to drag texts/objects around).
Question Type ID: Int
The question type ID contains an integer for the question type. This ID is the primary key for

this table.
<u>Questions table</u>
Topic ID: Int
QuestionType ID: Int
Question ID: Int
The question ID contains an integer for the question. It is primary key for this table and uniquely identifies each and every question.
Question Text: String
The question text is a string which stores the question which will be posed to the test taker.
Question Multimedia: String
Contains a string that specifies the directory of the multimedia location for the question (if it exists). This value will only filled if the question requires an image or video to be shown (e.g. showing positions of boats, a video of ship lights moving in the fog). In cases where multiple usage of multimedia is required, all the file locations will be concatenated together and delimited by ";"
Option 1 (Correct Answer), Option 2, Option 3, Option 4 : String
Option 1 to 4 contains the available options for any question with option 1 being stored as the correct answer. If the options are imaged-based, they will hold a string which specifies the directory of the image location for the options.
Explanation 1, Explanation 2, Explanation 3, Explanation 4 : String
Each contains an explanation as to why that answer is right or wrong with reference to the actual question. The explanation could also be a string that specifies the directory of where

the imagerial explanation is (if it exists).

6.3. Security Issues

6.3.1. Client-side

Our client behaves similarly to a standalone application, only updating its files and data from the server when necessary. As such, there are no major security flaws that may harm other test takers or the test taker himself.

There is, however, the possibility of an attack on the test taker's mobile device, resulting in the tampering of test taker's files and subsequent loss of the test taker's progression. To combat this problem, we will store a backup of the test taker's files in a separate location on the phone. An extension for future development would be to allow test takers to back up their progression onto our server and download back into their device when necessary. This extension would also provide test takers with the flexibility of changing the device they run our application on.

6.3.2. Server-side

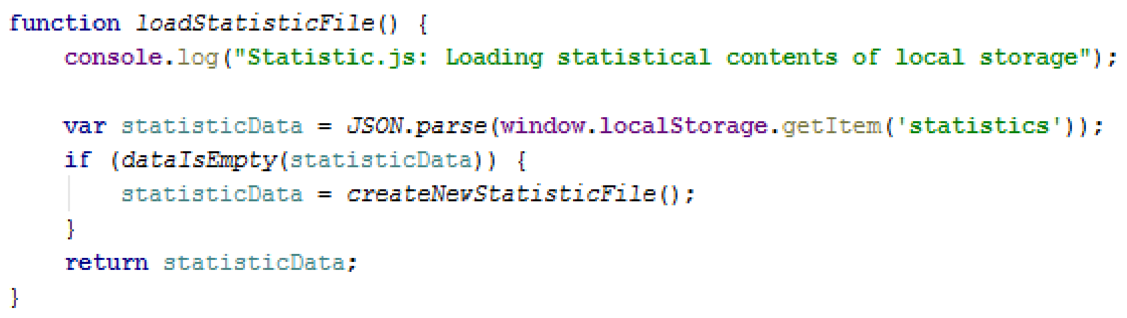
Being a typical web application, it is prone to the various common web application security risks. These include cross-site request forgeries (CSRF), SQL injections and cross-site scripting. These flaws, being common across most web applications, are easy to tackle with conscientious development. For example, to prevent CSRF, we can include a hidden CSRF token where there is form input. SQL injections can be avoided through parameterized queries.

7. Software Engineering Practices and Implementations

7.1. Robust Code

7.1.1. Handling null pointers

JavaScript uses the value `null` and `undefined` to specify unassigned or empty variables. To make our code robust, we handle such cases of `null` and `undefined` so that BoatMate can continue to run without giving any error. To do so, variables are checked with `isEmpty()` functions to ensure that the content returned by other functions is correct before processing them. An example is shown in the code snippet below:



```
function loadStatisticFile() {
    console.log("Statistic.js: Loading statistical contents of local storage");

    var statisticData = JSON.parse(window.localStorage.getItem('statistics'));
    if (dataIsEmpty(statisticData)) {
        statisticData = createNewStatisticFile();
    }
    return statisticData;
}
```

Fig 7.0 - Handling null pointers

7.1.2. Using Names for Constant Numbers

We define constant numbers or values to allow for easy adjustments of any constant values.

```
BoatMate.Navigation = function (game) {
      this.CAM_SPAN_WIDTH = 2000;
      this.CAM_SPAN_HEIGHT = 2000;
      this.TILE_WIDTH = 64;
      this.mapSize = 0;
      this.THRESHOLD = 5;
      …
}
```

7.2 Secure Code

As the content of our local question database is compiled from various trusted websites, there are no potential threats or risks of having data being stolen. As such, there is no need to secure the database. However, should any maritime academy wish to acquire the project and add their own questions, future developers might need to consider encrypting the database or moving the local database to the server side.

There is also no login system that needs to be secured. As such, in the current implementation, we put more focus on preventing buffer overflow since we have many assets to handle.

7.3 Clean Code

7.3.1. Asset Preloading

In making BoatMate, we have many scenes which are handled by one JavaScript file each. To make each scene concise, all assets (i.e. images, sounds, etc.) in a scene will be preloaded by the file that runs the scene. Exception will be given to universal assets such as inventory and menu assets.

7.3.2. File Categorization

Code written for Boatmate is split such that each JavaScript file only contains code that is relevant to a particular feature of Boatmate. For example, each mini game is separated into its own JavaScript file and each state of the game is in its own JavaScript file as well.

7.3.3. Naming Conventions

For each scene description files (in JSON format), all filenames start with the island name, followed by the scene name. For example, for island Emergency, there can be EmergencyCenter.json, EmergencySouth.json and EmergencySouthMcq.json for the center of island scene, south of island scene and the MCQ stage at the south of island scene respectively.
In terms of variable names, all variable will use camel casing, e.g. 'likeThis'. Variables in a scene file (for example the code file for island scene or mcq) may have description for its object type

in the name. For example, sprites loaded into the scene will have the name ‘spriteNameImage’ since they will be loaded as image with many extra properties that we can define.

7.3.4. Using names for constant numbers

JavaScript is a weakly-typed language, thus it has no concept of enum. Instead, constant numbers or values are assigned context-driven string values so as to improve readability whenever appropriate. For example, we use an enumerated type to better categorize our constants.

```
this.scenarios = {
      SUCCESS: 0,
      FAILED_NO_TIME: 1,
      FAILED_COLLIDED: 2,
      FAILED_WRONG_SIDE_BOAT: 3
};
```

That being said, this is not implemented universally as there are enum types that need to be serialized to and deserialized from JSON and implementation of such will follow https://stijndewitt.wordpress.com/2014/01/26/enums-in-javascript/, where although there will be repetition of words, it can be safely serialized and deserialized.

7.3.5. Comments

Comments will be used to describe code execution flow. This is useful as using both Phaser framework and JavaScript allows addition of variables into object without any formality of classing. Comments will be used to supplement and describe what happens and why it is needed.

8. Future improvements

8.1. Extending to Other Maritime Courses

As Boatmate is designed to be extensible, we may look into the possibility of adding more maritime courses in the future such as Advanced Powered Pleasure Craft Driving License (APPCDL) and Lifeboat Fall Prevention Device (FPD).

This extension can easily be done as game content managers are able to add questions that are relevant to specific courses into the databases under different course tags. Furthermore, they are also able to create game levels accordingly with the level editors using the relevant questions.

8.2. Syncing Game Save Across Different Devices

There are currently no systems in place for test takers to load their game progress across different devices. Therefore, future improvements include allowing test takers to sync or load their game progress and statistical data across different devices. This removes the restriction of users having to start a new game or maintain different game progresses between playing Boatmate on the computer and mobile phones/tablets.

This can be easily achieved by creating a new component in the client to upload a client's game state information to the server and storing it there. A new table in the database would have to be created and user accounts would have to be managed as well.

8.3. Meta-goals and Game Recognitions

Further improvements to Boatmate may also include the addition of achievements, trophies, badges or challenges. These secondary goals provides test takers with the impetus to explore the game more and motivates them to play the game and learn more by unlocking these goals. They can also unlock cosmetics for the ship by achieving these goals. Examples of achievements may include passing a level without any mistakes or playing the game for 30 consecutive days.

As gameplay statistics pertaining to the test taker's answering of questions are already stored, for goals involving the answering of questions, It can be easily used to determine if a test taker has met the requisites for a particular goal. In terms of other gameplay goals such as unlocking islands etc, it can be determined from the test taker's game state.

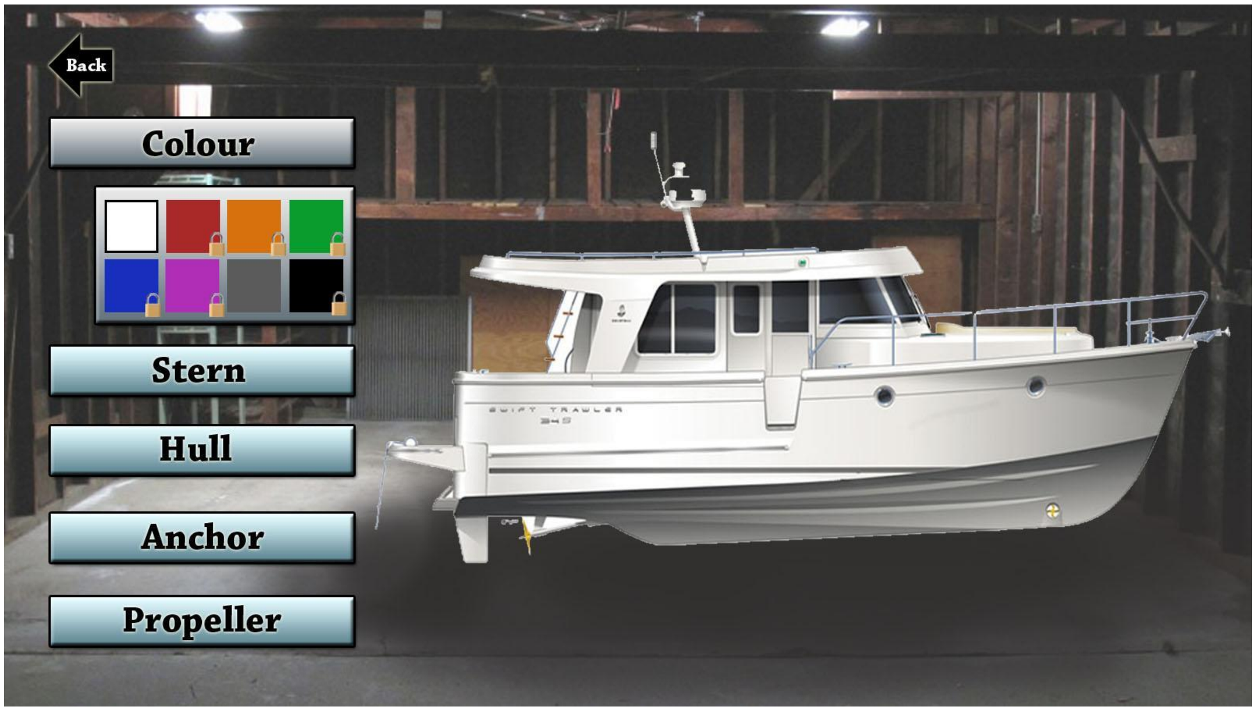
8.4. Statistics Collection by Game Content Manager

The server can be improved to allow sending of statistics from clients to the server. This would allow the game content managers to receive feedback on how well people are answering the questions in the game. It also can provide feedback as to which topics people are more interested in doing. In combination with syncing test taker's game state across multiple devices, the game content manager can also know who is having difficulty with which topics and consider approaching the individual directly to ensure he has the concepts right.

This implementation requires the creation of a component in the client in order to update a test taker's statistics into the server, which is easily achievable. The server also is required to be modified to accept the statistics and store them in a statistics database.

8.5. Customization

It refers to cosmetic customization of current watercraft. Such customization will not affect gameplay or game progression and only affect the visuals of the game.



Mockup of how test takers can customize their watercraft

Test takers can unlock and receive cosmetic upgrades which they may use to improve the look and feel of the boat they are controlling in the navigation screen. These cosmetic upgrades serve as collectibles or a form of encouragement and personalization by test takers who want to collect them all.

This can be done by creating a separate file on the client side to store the test taker's chosen customisations and unlocked customisations. The method of unlocking customisations can be similar to that of meta-goals mentioned above.

8.6. Adaptive Targeted Learning

As an extension to what Boatmate will achieve in its targeted learning, through the collection of statistics of the user's gameplay, we can explore the possibility of responsive learning. The game can choose the type of gameplay for the user, dynamically altering the user's experience of the game in order to fit his preference.

This can be done by referencing the statistics stored on the test taker's client. For example, Boatmate's user statistics may show that the test taker has higher test scores for questions that appear in an animation. Boatmate can thus adapt and provide more questions that come in the form of an animation to the test taker.

9. Evaluation

9.1. Evaluation on User Requirements

Captain Fadil, a PPCDL instructor, was consulted and a demonstration of our game prototype was introduced to him using a laptop. An evaluation was performed based on user requirements highlighted in Section 2. However, not all user requirements have been evaluated as the prototype has not implemented all features.

9.1.1. Evaluation of Test Taker Requirements

Criteria	Evaluation/Feedback
Ease of use (Test takers can use Boatmate conveniently)	**Observation**: ● Mouse input proves to be intuitive and the user did not need additional instructions to navigate the interface. ● Instructions given during individual game sections, such as MCQ quiz, is clear to the user as he is able to achieve objectives without guidance. ● Visual feedback is insufficient as the user is unaware of which areas are interactable. ● Keyboard controls may be slightly confusing due to initial fumbling of controls while playing the random event. **Feedback**: ● User finds the system straightforward and easy to use. ● User agreed that multi-platform support will be useful.
	Evaluation: Text feedback seems to be sufficient, however, visual cues will need working on. The team may change some GUI layouts to ensure clarity for test takers.
Educational effectiveness (Test takers are able to learn and practice their PPCDL concepts using Boatmate)	**Feedback:** ● It is beneficial that the MCQ style of questioning fits the way students are assessed in theory tests. ● Education on practical test can be improved. ● Emphasis on common student mistakes during both theory tests and practicals should be made. ● Some content is lacking, such as practical concepts and collision regulations. ● Existing content may be lacking details, such as the boat fire

	extinguisher having to be of foam type instead of any other type. ● The user suggests that the oral component of PPCDL can also be incorporated into Boatmate. ● It will be useful to integrate Singapore's map into Boatmate as many test takers are not familiar with the surrounding locations.
	Evaluation: The team needs to work closely with the PPCDL syllabus and consult Captain Fadil to ensure that our content is as accurate as possible. Certain topics in PPCDL will have more coverage than others due to their complexity or higher percentage of failures based on Captain Fadil's advice.
Progress Tracking (Test takers can track their progress)	**Feedback:** ● Current PPCDL courses do not explicitly keep track of the students' progress; the progress is kept in check by the students or instructors themselves. This will be very useful for students, especially when self-learning.
	Evaluation: Seeing its importance for test takers, the team will focus on this feature once all high priority features have been implemented.

9.1.2. Evaluation of Game Content Manager Requirements

Proper evaluations on Boatmate's level editor system were not conducted as a prototype was not available prior to the meeting with Captain Fadil. However, Captain Fadil has mentioned a few key points which we find provides substantiation for the implementation of a level editor.

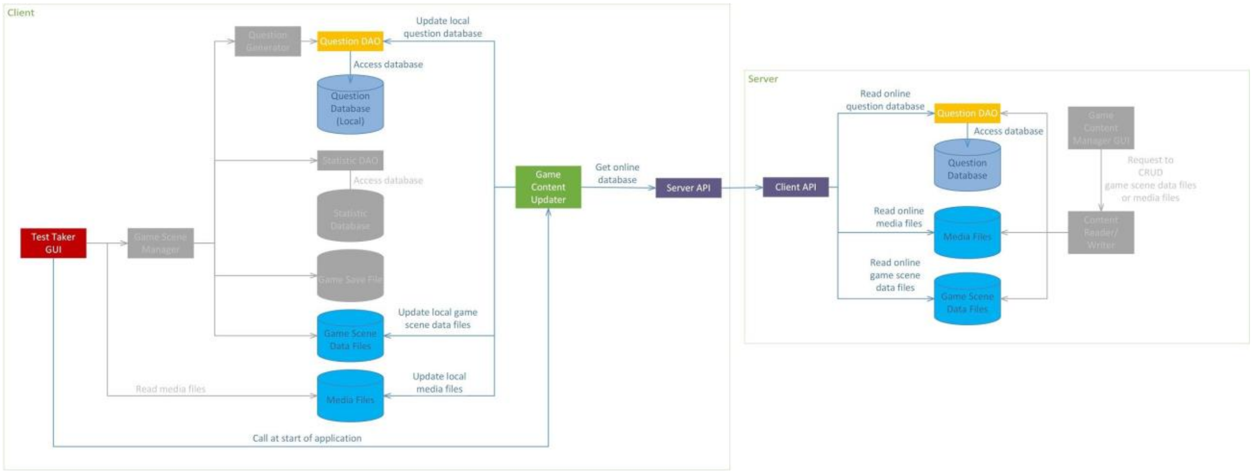
- Captain Fadil has displayed a keen interest in spreading awareness about the Lifeboat Fall Prevention Device (FPD). As FPD is a course similar to PPCDL, we find that the ability to easily incorporate additional learning material can greatly enrich Boatmate.
- International boating laws may or may not overlap with local laws. Thus it will be useful if Boatmate can change some of its content depending on the location of the user.

Thus, Boatmate's content management and level editor system may meet the content manager's user requirements.

9.2. Evaluation on System Requirements - System Walkthrough

In this section, we will walk through our architecture to prove that our architecture has allowed us to achieve system requirements of Boatmate. Notably, we will evaluate update of game content and playing the game from top layer gameplay - i.e. map navigation - down to the bottom layer of MCQ gameplay until test taker finishes the gameplay.

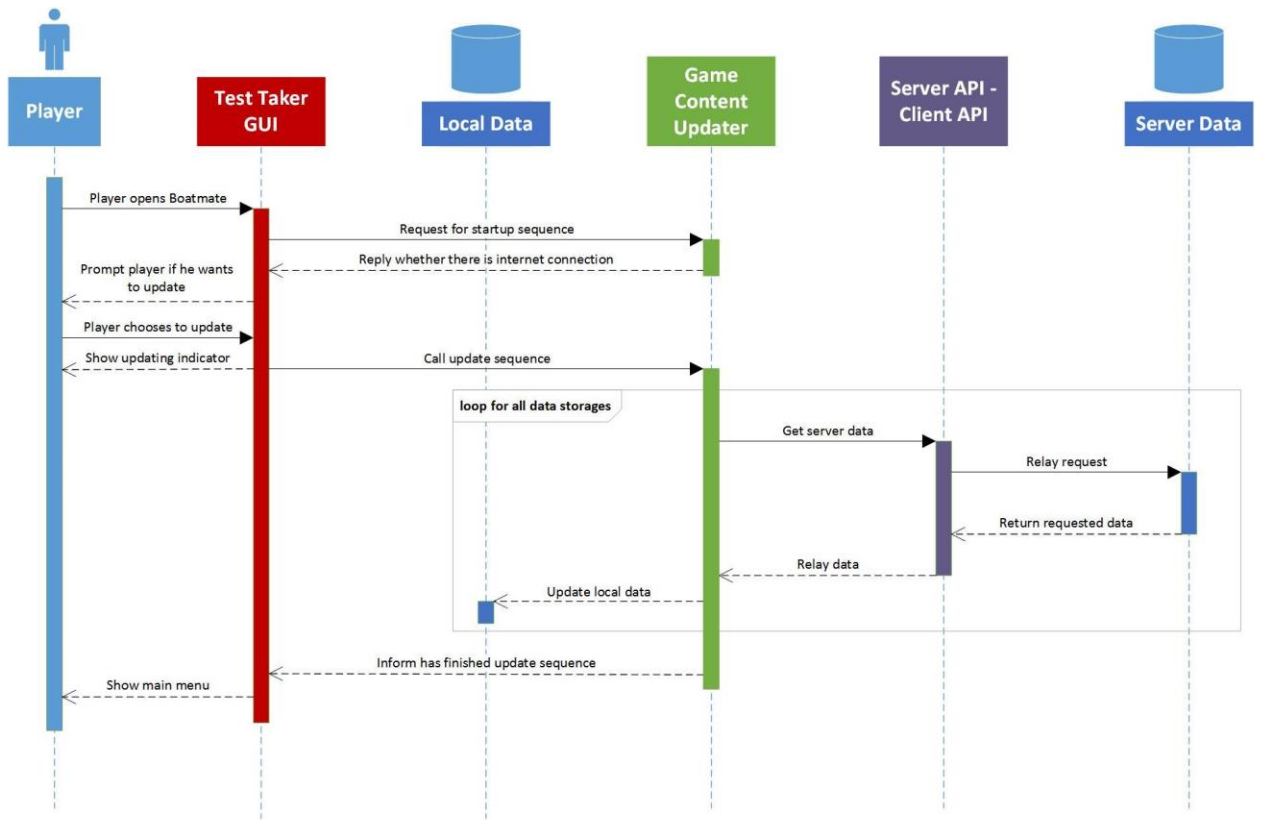
9.2.1. Updating Game Content at Start of Application



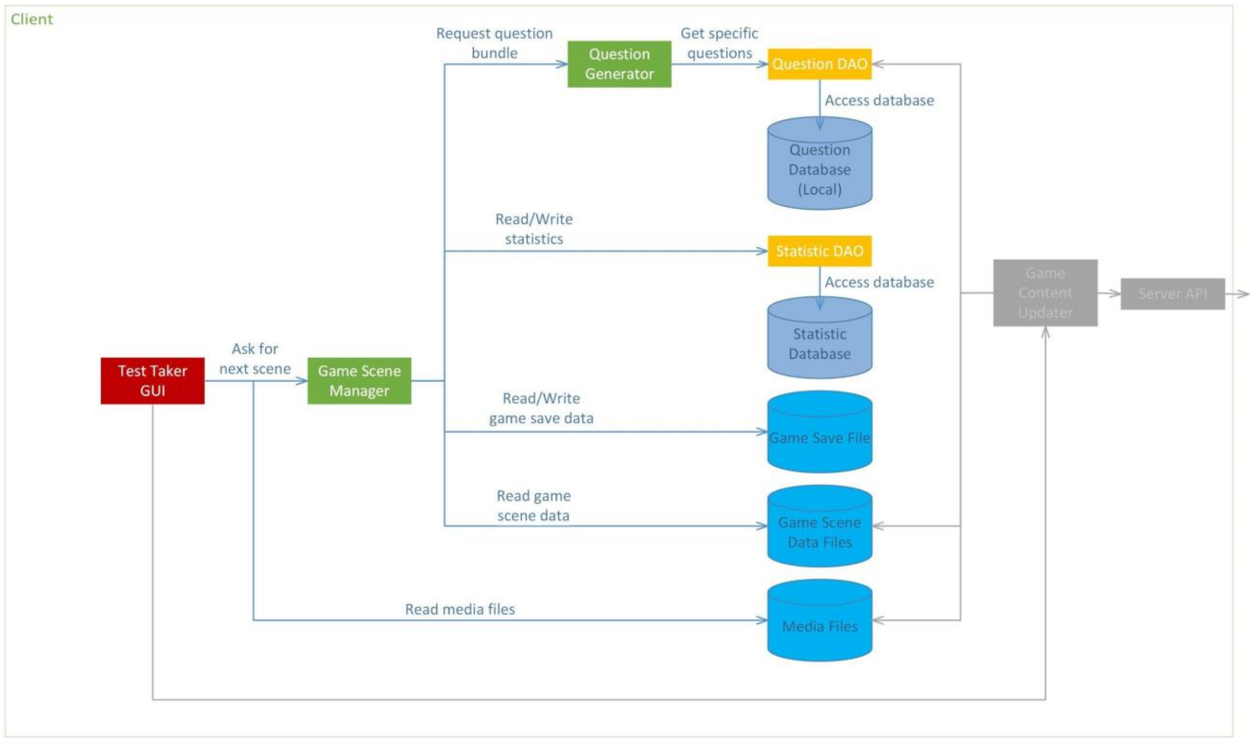
The diagram above shows an overall view of the components involved when updating the game content at the start of the Boatmate application. Uninvolved components are colored gray.

Below is a sequence diagram of how the highlighted components interact in updating the local game content. The diagram is drawn on these conditions:

- Internet connection is present, thus Test Taker GUI will act as if Game Content Updater reply that internet connection is present.
- Since Server API and Client API are abstracted interface to help relaying data, they are drawn as one to assuming perfect communication between the two to keep the sequence diagram simple.
- The various files and databases is represented by one Data as representative, one for local and one server. Game Content Updater will loop through the different files and databases.


Player
Test Taker GUI
Local Data
Game Content Updater
Server API - Client API
Server Data
Player opens Boatmate
Request for startup sequence
Reply whether there is internet connection
Prompt player if he wants to update
Player chooses to update
Show updating indicator
Call update sequence
loop for all data storages
Get server data
Relay request
Return requested data
Relay data
Update local data
Inform has finished update sequence
Show main menu

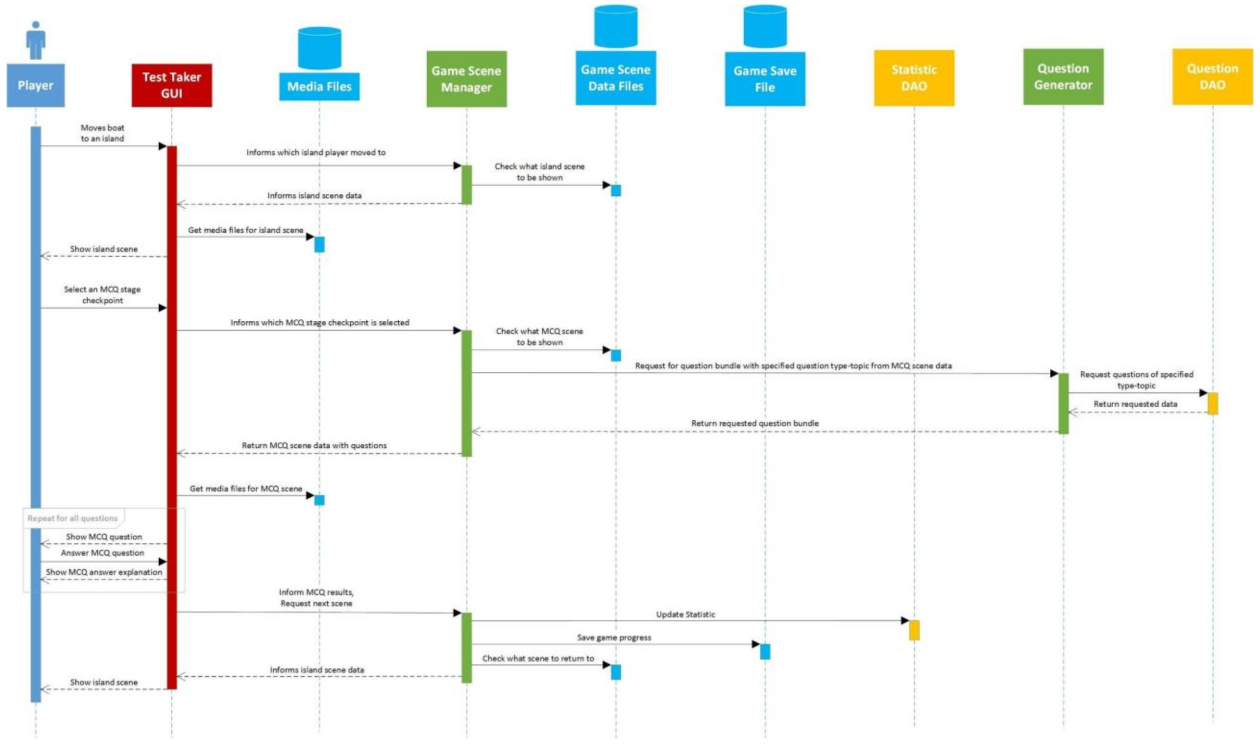
9.2.2. Playing an MCQ Level from Navigation Gameplay



Above is a highlight of the components in the system architecture which are involved while playing Boatmate from the navigation scene to the end of an MCQ scene. Uninvolved components are colored gray. The server side is not drawn as it is not involved in the gameplay, proving that Boatmate can run offline.

Below is a sequence diagram of how the highlighted components interact in playing Boatmate from the navigation scene to MCQ scene. The diagram is drawn on these conditions:

- Only DAOs are drawn. The databases are not shown to keep the diagram compact, especially that the DAOs only relay the database request, thus it is a straightforward interaction.


Player
Test Taker GUI
Media Files
Game Scene Manager
Game Scene Data Files
Game Save File
Statistic DAO
Question Generator
Question DAO
Moves boat to an island
Informs which island player moved to
Check what island scene to be shown
Informs island scene data
Get media files for island scene
Show island scene
Select an MCQ stage checkpoint
Informs which MCQ stage checkpoint is selected
Check what MCQ scene to be shown
Request for question bundle with specified question type-topic from MCQ scene data
Request questions of specified type-topic
Return requested data
Return requested question bundle
Return MCQ scene data with questions
Get media files for MCQ scene
Repeat for all questions
Show MCQ question
Answer MCQ question
Show MCQ answer explanation
Inform MCQ results, Request next scene
Update Statistic
Save game progress
Check what scene to return to
Informs island scene data
Show island scene

We have run through our prototype with regards to this scenario, and the relevant components are working as intended in the sequence diagram shown above.

10. Team Achievement

10.1. Meeting User Requirements

Based on our user evaluations, we find that Boatmate meets most of the basic requirements of a PPCDL test taker as well as a game content manager. Many of Boatmate's possible extensions also address both users' supplementary requirements. We will continue to work on Boatmate's basic features and seek evaluations from our users in order to cater Boatmate to their requirements.

10.2. Technical Challenges and Lessons Learnt

We are currently using frameworks such as Phaser as the game engine and CocoonJS as the packager for multi-platform applications. However, we find that Phaser and CocoonJS are not 100% compatible. Some functions in Phaser do not work as intended in CocoonJS, such as the lack of support for Phaser's multi-line text in CocoonJS. Therefore, intensive research has to be done for additional frameworks that we may choose to incorporate into our project to prevent similar experiences in the future.

Using HTML5 means that we have to use JavaScript as our programming language. The problem is that JavaScript is more of a scripting language, which makes software design considerations more complicated than using other Object-Oriented Language such as Java. Currently, in our prototype, we have attempted to apply object oriented principles onto our JavaScript code as well as separate our code into different files based on their functionalities. However, we still need to further refine and refactor what has been coded to better adhere to our software components design.

11. Individual Achievements

For each team member, indicate which parts of the system design are under his or her responsibility.
Highlight any technical challenges and lessons learned, if applicable.

In this semester, in the development of the latest prototype, all members of the group were focused on the GUI and generation of game scenes for testing purposes. We focused on this as it was crucial for our user evaluation to ensure we were on the right track. Details of who implemented which portions of the prototype are elaborated on below.

11.1. Terence Rei Jie Then (A0094682U)

Implemented the drag and drop stage for sorting out the right items that should be in the watercraft. Made the user interface overlay to display the test taker's existing inventory. Technical challenges faced were mainly figuring out how Phaser worked and what available functionality there was to utilise. For example, there was no function to allow an overlay of UI over the game. This therefore had to be done by creating a separate class accessible by all game states and calling the function at the end of every state.

11.2. Hans Adrian (A0100032M)

My responsibility covers MCQ stage and moving towards Navigation gameplay. The challenge I met was to make text works properly in mobile version, because Phaser's text has problems with CocoonJS packager for Canvas+ which makes texts displayed incorrectly. Currently we are using less efficient packager from CocoonJS to make texts displayed correctly. It seems that working with multiple frameworks requires careful evaluation before hand, so that we know everything will work on developing all required features.

11.3. Christopher Andy Weidya (A0099568A)

The areas under my responsibility include the implementing map navigation as well as generating in-game graphics. While implementing the initial stages of map navigation,the interaction between converting screen coordinates to isometric coordinates and the shifting of isometric elements prove to be both a mathematical and technical challenge. Through this process, my understanding of isometric projection has improved significantly. Furthermore, I also learnt how to create graphics that suit the isometric map which Phaser generates after repeated experimentations and graphic modifications.

11.4. Ko Wan Ling (A0100729M)

I implemented the man overboard mini game that occurs randomly while travelling in the navigation stage. This proved to be a challenge as it was essentially simulating the practical assessment of rescuing a man overboard, while taking into account the wind direction and water currents. Furthermore, there was an issue with Phaser's collision detection, which was crucial in detecting if the player controlled watercraft has collided with the man. Resolving this issue via implementing a collision system without physics interactions has allowed me to learn much about collision detection in games. In addition, I have also helped in producing in-game graphics, in particular, UI elements.

11.5. Tan Zheng Jie Matthew (A0101810A)

My responsibilities include the making of the Question Generator, Question DAO and the stage level for the first island which tests the test taker on basic seamanship. One technical challenge was to accurately track and update the test taker's progress throughout the island exploration and traversing between game components. Therefore, careful planning needs to be done to prevent the looping of progression. It might even be useful to draw progression charts to prevent users from unlocking unintended progression.

12. Acknowledgements

Boatmate would like to thank the following people for their continuous and unwavering support throughout the semester:

Leow Wee Kheng - Associate Professor, School of Computing, National University of Singapore

Stephane Bressan - Associate Professor, School of Computing, National University of Singapore

Mohd Fadil Bin Yunos - PPCDL course coordinator, Singapore Maritime Academy